"Spending" Efficiency to Go Faster

Dr. Alistair Cockburn
Humans and Technology

Have you ever been on a project where some person or group is holding up the works? They are called the "bottleneck" station, and here are some usual and unusual strategies for improving output in the presence of various bottlenecks.

In any project or organization, some person, group, or station inevitably acts as a bottleneck to the organization's output. This is, of course, trivially true: Once the output of that station improves so it is not the bottleneck, some other station becomes the limiting factor.

For just that reason, I will not discuss in this article options about improving the performance at individual stations. I will, rather, discuss ways to improve total system results once you have tried all you can think of for the key stations. If you find a way to improve the performance at the bottleneck station, then you get to start all over, working out where the new bottleneck is and how to improve total system performance in the presence of that bottleneck.

Assuming, then, that you have done all you can to improve the output ability at the bottleneck station, it is sometimes possible to further improve output by putting attention on the non-bottleneck stations.

The odd part about these strategies is that when we use the spare capacity at the non-bottleneck stations, we will sometimes deliberately allow "rework" in order to gain an advantage at the bottleneck station. This is counterintuitive to most people: Most of our industry is founded on the notion that we should avoid rework like the plague.

I will refer to this as "spending" efficiency locally for a global gain.

Once you start looking for this, you will see people in ordinary life doing exactly that: Those who have a bit of spare time find ways to help the bottleneck group and streamline the overall flow. The way in which efficiency is best spent differs according to the situation. Taking a look at those alternatives is what this article is about.

A Sketch of the Argument

If you optimize each independent operation in a chain of activities, you are quite likely to not get the best total output from the entire chain. This has been studied and documented for years. It is, among other things, the basis for Eliyahu M. Goldratt's Theory of Constraints [1, 2].

The usual advice is to find the bottleneck station, make it work better until it is no longer the bottleneck, and then pay attention to the new bottleneck station.

While that advice is, of course, correct, it is lacking in two regards:

- For any organization, there is eventually a point when the workers and managers are working at their limit. They may not be able to hire any more people at the bottleneck station or find any other way to improve that group's productivity. This point will be reached in all cases, whether temporarily or due to fundamental limits.

"Those who have a bit of spare time find ways to help the bottleneck group and streamline the overall flow. The way in which efficiency is best spent differs according to the situation."

- There might be other things other people can do to improve the output of the system as a whole.

Here is a simple example taken from ordinary life: John was folding brochures for a part-time job. Sean was doing some other work nearby and wanted to find a way to help, but in this situation couldn't help fold brochures. Sean found that he could help by leaning over from time to time and tilting the stack of papers so John could pull new ones off the top faster.

The point to observe here is that Sean didn't do John's work for him, but still found a way to speed up John's work.

The basis for these sorts of strategies is another truism: People at the non-bottleneck stations have spare capacity. By using that spare capacity in interesting ways, we can sometimes improve the total system output.

Basic Alternatives

When I first went looking for suggestions as to what people should do with their extra time when working at non-bottleneck stations, I found very little.

The first suggestion, given by Goldratt in the context of the Theory of Constraints, is that the people "sit on their hands" so they don't silently turn into bottlenecks without noticing it:

> During a presentation of the five steps of focusing, I can recall Eli saying that at non-bottlenecks people should sit on their hands until there is work to be done. When there is work to be done, then they should work as fast as they can and then return to sitting on their hands. But this act of subordination is extremely difficult as it goes against common practice of maximizing the utilization of every resource, hence the new rules, "Utilization and Activation are not synonymous" and "The level of utilization of a non-bottleneck is not determined by its own potential but by some other constraint in the system." [3]

Goldratt also said:

> ... we have to face the fact that this conclusion means that under no circumstances should we release materials just to supply work to workers ... the worker is not running the machine. He is standing idle. [4]

The Agile and Lean manufacturing communities do one better [5, 6]. They cross-train people at adjacent stations so that if one of them becomes overloaded, the neighbor can do some of his work until the bottleneck moves. These two suggestions—sit idle and do some of the work of the bottleneck station—are good but not always applicable, appropriate, or optimal.

I will develop two more strategies in the following sections:

1. *Simplify* the work of the bottleneck station (as with tilting the stack of papers).
2. Let non-bottleneck people *rework* their ideas to reduce future rework or speed decisions at the bottleneck station.

The second of these is the least obvious and therefore most interesting. I'll illustrate both with case studies from software development projects.

Simplify the Work of Others

The first strategy is based on the company eBucks, a spin-off from a larger bank to create online rewards systems.

The eBucks case has been described in some detail in [7]. Here, I only present details relevant to the current topic: what to do at non-bottleneck stations. I'll break the story into three parts:

1. Organizational structure.
2. Development methodology.
3. Changed strategy.

Organizational Structure

eBucks had about 50 employees at the time, with three programmers who knew the domain and technology well and about 16 programmers fresh from college who were new to both the technology and the business. More such programmers were being hired at the time.

There were four business experts and two expert IT analysts who created and documented new initiatives for the programming team to develop.

Everyone sat within a few dozen steps of each other, in accordance with Agile principles [7]. The company released any new system features they had to the Web every two weeks. The requirements evolved in parallel with development while the programmers were programming.

The company was gaining market dominance in part because they deployed new functionality to the public faster than their competitors could match.

Development Methodology

The four market specialists, in dialogue with their external contacts and with the help of the two experienced IT analysts, would draft function requests (*initiatives*) for the development group to implement.

At the time, there were about 70 initiatives in the queue. Given the proximity of seating and the fact that they were deploying to the Web every other week, the most obvious strategy to consider would be to increase verbal communication between the analysts and the programmers, as is recommended in the standard Agile literature [7].

Closer investigation indicated that this would be a mistake. The problem was that the programmers were overloaded with requests. With four times as many initiatives as programmers, each programmer was working on between four and six initiatives at one time. The programmers were mostly inexperienced, fresh from school, and did not yet understand the problem domain very well. Because of their newness and because of the wide range of assignments in play at any one time, the programmers could not keep the details of their assignments in their heads.

Changed Strategy

The first change, of course, was to reduce the number of initiatives in play, so that each programmer was working on one (or at most two) initiatives in any one week. This was still not enough, given their newness to the domain and the enormous backlog of initiatives they were facing.

> *"The problem was that the programmers were overloaded with requests. With four times as many initiatives as programmers, each programmer was working on between four and six initiatives at one time."*

In the context of this article, the programmers were quite severely the bottleneck station; adding more programmers wouldn't help, nor could the existing ones program faster. The only place found to improve the output of the programmers was to ensure that they were not doing *accidental* rework due to forgetting what the domain rules were, or spending time doing domain research when other people were available to do that. The goal was to keep the programmers programming usefully.

Consequently, we went against the standard advice of the Agile literature and, rather than using verbal communication to pass along the evolving requirements, we agreed that the market specialists and business analysts would write down for the programmers fairly detailed use cases, business rules, and data descriptions.

The analysts then walked the programmers through the text they had written and left the text for the programmers to refer to as they worked.

The result was that the programmers could work in an uninterrupted, *heads down* mode for most of the day, instead of stopping to ask questions or do research.

Lest this seem like the natural, default, or standard solution, it should be reiterated that the reference Agile material on Crystal [7], Extreme Programming [8], and Scrum [9] all recommend reducing written material and increasing verbal material.

In the next project, we did indeed reduce the written material given to the programmers; in that project, however, the programmers were not the bottleneck.

Rework Strategically

The second strategy is based on a medium-sized IT project I call Winifred. The project was a success in the following sense: The team delivered the contracted functionality on time, in three-month increments; the system solved the problem that management was concerned about; the users utilized it as it rolled out each quarter; and the system is still in active use and maintenance 10 years later. Project Winifred has been described in some detail in [10]. Here, I only present details relevant to the current topic: what to do at non-bottleneck stations. I'll break the story into the same three parts as for eBucks.

Organizational Structure

The project was fixed-scope (240 use cases), fixed-price ($15 million), and fixed time (18 months), using several technologies: COBOL, Smalltalk, and relational databases for the production data. It was staffed with 24 programmers in a total team of 45 people (at its peak). It was delivered incrementally to the user base every three months.

The project used technologies in an architecture common in the 1990s: A Smalltalk client on a PC was connected to a server running a relational database that was connected to the company's mainframe system which ran programs written in COBOL. All three technologies—COBOL on the mainframe, relational database on the server, and object-oriented code in Smalltalk on the client's PC—were within the project.

The contractors sat on the same floor and worked closely with the contracting

company's employees. Programmers sat in two or three team rooms; the business analysts had offices several dozen steps away.

The agreement between companies allowed the users to change their minds about what they wanted within each three-month development period. The limit put on this was that within the first six or seven weeks of the 13-week period, the users could change any requirement in any way and could refine their requirements up to week eight or nine. Because testing and deployment preparation work took four to five weeks, there was very little time between when the last requirements changed and when the system was given to the production team.

Two or three Smalltalk programmers were attached to each business analyst in a function team.

The difficulty was that there were only two database analysts (DBAs) for all four function teams. The DBAs quickly became the bottleneck station because they couldn't revise the database fast enough to match the requirements and design changes.

Development Methodology

A detailed description of the methodology can be found in [10]. What is important here are the linkages between the business analysts (BAs) and the programmers, and between the programmers and the DBAs.

The BAs met with user representatives each week to discuss the current release's requirements and to show the progress being made. The BAs documented their meetings with the users only for tracking purposes; they conveyed detailed information to the programmers verbally as needed, daily and after each meeting. Because of the high quality of verbal communication, the BAs were saved from having to revise detailed requirements documents with the frequently changing information.

The programmers attached to each BA changed the domain model as needed from week to week, to keep up with the requirements changes, and also to improve the design.

It soon became clear that the DBAs could not keep up with those changes.

Changed Strategy

The strategies we considered might be enumerated as follows:

1. Wait until requirements settle.
2. Get rid of some programmers.
3. Hire more DBAs.
4. Make the DBAs keep up with the programmers' changes.
5. Have the programmers stop making changes to their design.
6. Let the programmers create trial designs.

The first was not allowed under the terms of the agreement; the second would not speed the project; the third, for reasons of domain knowledge and corporate information security, was not allowed; the fourth was not possible in practice; and the fifth would produce an inferior design. That left the least obvious sixth choice.

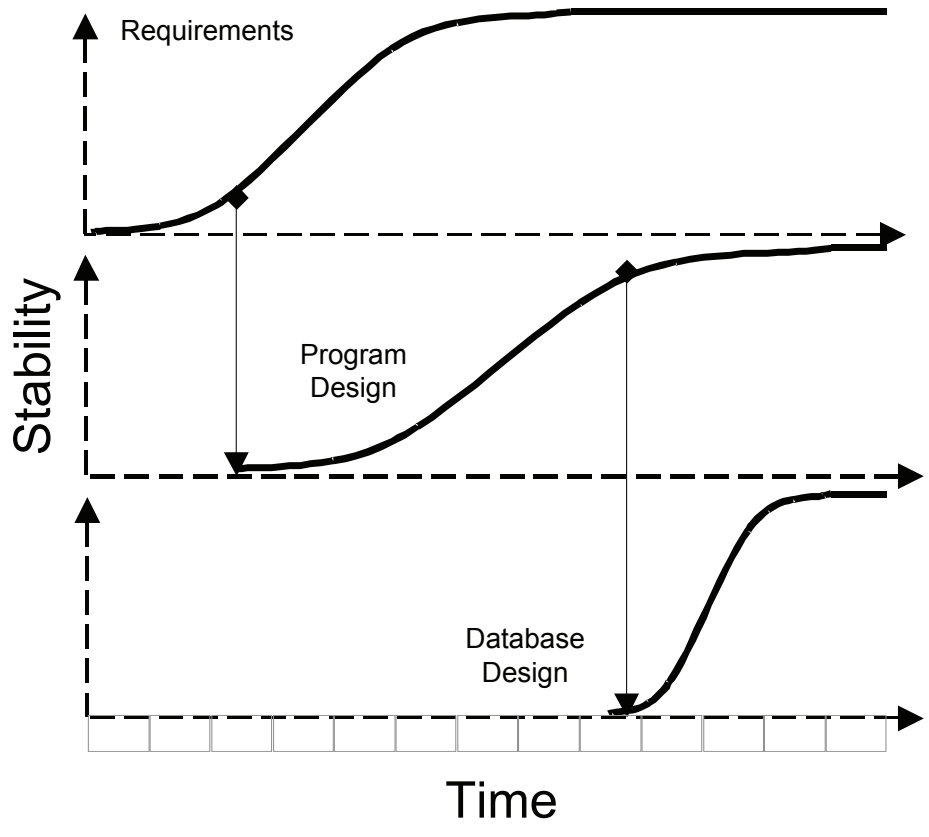
Since there were many more programmers than DBAs (and possibly because Smalltalk is so malleable an environment), the programmers had the capacity to experiment and improve the characteristics of the domain design without impacting the database design schedule. Allowing them to do this could end up speeding the database design work for the simple reason that there would be less design rework later.

Consequently, the programmers did not give the domain model over for database implementation until they had tried several iterations of their design, could assert that it was "relatively stable," and had passed it through a design review with domain experts and the two DBAs.

In terms of what to do with excess capacity at a non-bottleneck station, there is a strategy different from sitting idle, doing the work of the bottleneck, or simplifying the work at the bottleneck; it is to use the excess capacity to rework the ideas to get them more stable so that less rework is needed later at the bottleneck station.

Normally, rework is considered "waste" and minimized, but here we see it used in a strategic manner.

Figure 1: *Project Winifred's Stations Triggered Differently Based on the Stability of the Upstream Material*



Analysis

Four strategies have been named so far for what people might do at a non-bottleneck station:

1. Sit idle.
2. Do the work of the bottleneck station.
3. Simplify the work at the bottleneck station.
4. Rework material to reduce future rework required at the bottleneck station.

In each of these, the efficiency of the non-bottleneck station is lowered, or "spent." This is what I meant in the title, about "spending" efficiency to go faster.

The ways in which the efficiency should be spent is an interesting and meaningful topic for any team, and should be considered deliberately and strategically for the situation at hand.

Rework as a Deliberate Strategy

The rework strategy is the least obvious strategy and deserves some more analysis. To understand its use in Project Winifred, consider the growing stability of the material at the three stations. At the start of each cycle, the requirements were unstable. They became more stable over the first eight weeks of each three-month cycle as they also became more complete.

The domain model was also unstable at the start of each cycle. It became also more stable and complete over time, but only had to reach the top level during the testing period (that is, around week 12 of the cycle). The database design had to reach the point of being stable and complete at the same time as the program. Figure 1 (adapted from [7]) shows how stability grew over time for requirements, program design, and database design.

Recognizing that the programmers had time to rework their designs and that the DBAs really only had one good shot at their design, it was arranged for the programmers to start their work from relatively less complete and stable input, while the DBAs would start their work from relatively more stable input.

The moment of transfer is shown in Figure 1 with the vertical arrow from the upstream station's material down to the downstream station's material. It was important that the people on the respective teams traded information continuously once the transfer had occurred, since the upstream material was changing while the downstream people were working from it.

The strategy of allowing rework is sensitive to the placement of the points of transfer shown in the figure. The fact that the DBAs needed stable information is reflected in the high position of the transfer point: They didn't have time to do rework and therefore they needed stable information. The fact that the programmers did have time to do rework is reflected in the low position of the transfer point and the longer time allocated for program design: They started sooner and reworked more.

If the programmers hadn't had time to revise their domain model, this would have been a poor strategy—the exact point I am trying to make with this analysis.

Downstream vs. Upstream Rework

In Project Winifred, it was the upstream station that had the extra capacity. The strategy we used was for those people to rework their design to improve its quality and to stabilize it before handing it on to the constrained downstream station.

Suppose, however, that it is the downstream station that has the extra capacity. This might happen, for example, when the marketing team is the bottleneck and can't decide which of several alternatives is preferable. Here, the downstream team might use their extra capacity to create several designs for the upstream people to choose from. The unused design would simply get discarded, another example of useful waste.

It is an interesting exercise to imagine the bottleneck station being at different places in the work stream, and working out what a useful rework strategy might be for the non-bottleneck stations.

Summary

Two software development cases and how they made different use of the extra capacity of their non-bottleneck resources were discussed (people whose work was not the speed-limiting factor in the overall output of the organization).

In the first case, eBucks, the non-bottleneck people did extra work to *simplify* the work of the people at the bottleneck stations.

In the second case, Project Winifred, the non-bottleneck people performed strategic rework in order to reduce the later rework of the bottleneck group.

In a thought experiment, we saw how a downstream team might create multiple designs for an upstream group to choose from.

When these strategies are added to the more commonly known ones—of having the non-bottleneck people sit idle or do the work of the bottleneck people—there are five strategies to choose from for how to make use of the "excess efficiency" available at non-bottleneck stations:

1. Have them sit idle.
2. Have them do the work of the bottleneck station.
3. Have them simplify the work at the bottleneck station.
4. Have them rework material to reduce future rework required at the bottleneck station.
5. Have them create multiple alternatives for the bottleneck station to choose from.

For each of these, the efficiency of the non-bottleneck station is strategically lowered; that is, the efficiency is "deliberately spent" in a particular way to gain an overall advantage in system output. The ways in which efficiency should be spent differs according to situation.

The examples in this article were all taken from software development. It should be clear that these ideas apply to organizations and projects in general.◆

References

1. Goldratt, Eliyahu M., and Jeff Cox. The Goal: A Process of Ongoing Improvement. Great Barrington, MA: North River Press, 2004.
2. Goldratt, Eliyahu M. Theory of Constraints. Great Barrington, MA: North River Press, 1999.
3. Bowles, Jim. From a posting on the theory-of-constraints experts mailing list. No URL available.
4. Goldratt, Eliyahu M., and Robert Fox. The Race. Great Barrington, MA: North River Press, 1986.
5. Reinertsen, Donald. Managing the Design Factory. The Free Press, 1997.
6. Personal discussion with Donald Reinertsen, Jan. 2005.
7. Cockburn, Alistair. Agile Software Development: The Cooperative Game. 2nd ed. Addison-Wesley, 2006.
8. Beck, Kent. Extreme Programming Explained: Embrace Change. 2nd ed. Addison-Wesley, 2005.
9. Schwaber, Ken, and Mike Beedle. Agile Software Development With Scrum. Upper Saddle River, NJ: Prentice-Hall, 2002.
10. Cockburn, Alistair. Surviving Object-Oriented Projects. Addison-Wesley Professional, 1998.

About the Author

Alistair Cockburn, Ph.D., is an expert on object-oriented (OO) design, software development methodologies, use cases, and project management. He is the author of "Agile Software Development," "Writing Effective Use Cases," and "Surviving OO Projects," and was one of the authors of the "Agile Development Manifesto." He defined an early Agile methodology for the IBM Consulting Group, served as special advisor to the Central Bank of Norway, and has worked for companies in several countries. More can be found online at <http://alistair.cockburn.us>.

1814 East Fort Douglas CIR
Salt Lake City, UT 84103
Phone: (801) 582-3162
E-mail: acockburn@aol.com